5. Security Services

ECS security architecture must meet the requirements for data integrity, availability, and confidentiality. ECS Security Services meets these requirements by incorporating a variety of mechanisms to establish and verify user accounts, issue and verify passwords, audit user activity, and verify and protect data transfer. Security logs will be monitored and security reports generated by the System Administrator as required. Several FREEWARE products provide tools for authentication and network and system monitoring: SATAN, Crack, anlpasswd, TCP Wrappers, Secure Shell, and Tripwire. The Open Software Foundation's Distributed Computing Environment (OSF/DCE) employs Kerberos for authenticating user requests for network services. (DCE administration tools a are discussed in Section 3 of this document.) The FREEWARE product, SATAN, monitors networks and finds system security vulnerabilities. Three FREEWARE products -Crack, anlpasswd and Secure Shell -provide additional password checking for local system and network access. To monitor and limit access to network services, ECS Security Services use TCP Wrappers. Tripwire monitors for intruders changes to files by detecting and flagging any changes. Security Services also supports detection of, reporting, and recovery from security breaches.

The following section defines step-by-step procedures for M&O personnel to run the Security Services tools. The procedures assume that the requester's application for a Security process has already been approved by DAAC Management. It is recommended that access to these tools be controlled through the **root access only**.

5.1 Monitoring Network Vulnerabilities

The Security Administrator Tool for Analyzing Networks (SATAN) is a testing and reporting tool that collects a variety of information about networked hosts. SATAN gathers information about specified hosts and networks by examining network services (for example, finger, NFS, NIS, ftp). SATAN also gathers general network information (network topology, network services run, types of hardware and software being used on the network). The data is used to point out system vulnerabilities. Data can be reported in a summary format. Problems are described briefly and pointers provided to patches or workarounds.

Periodically, the System Administrator will run SATAN as **root**. The procedures are provided below.

1 Make sure that **DISPLAY** is set on your workstation.

2 From **/usr/local/solaris/satan-1.1.1** type **./satan**.

3 From the **SATAN Control Panel**, select **SATAN Configuration Management**. Set all variables or use the default values.

4 Go back to the SATAN Control Panel.

5 From the **SATAN Control Panel**, select **SATAN Data Management**. Create the SATAN database if it does not exist. When you create the database for the first time, you will see a warning message concerning password disclosures. Take no action and continue. The database is stored as **satan-data** in the directory

/satan-1.1.1/results.

6 You will be notified when the SATAN finishes creating the database and scans the system (network or cluster) for vulnerabilities.

7 From this screen, you can click on "Continue with Reporting and Analysis" or you can return to the **SATAN Control Panel**, to make this selection. Select the reports that you want to review.

5.2 Ensuring Password Integrity

One aspect of system security is discretionary access control based on user passwords. Passwords should be so unique that they are virtually impenetrable to unauthorized users. Two products provide utilities to create effective password practices. "Crack" detects weak passwords that could be easily bypassed in a "batch" mode. Anlpassword enforces strong password rules as the user is changing their password.

Crack and anlpasswd provide comprehensive dictionary, which can shared. These "source" dictionaries provide lists of words, which if used, would create vulnerable passwords. You can add other dictionaries, for example, acronym lists, to eliminate commonly used terms from being used as passwords.

Both products are installed in a secure location, that has, **root access only**. Such precautions are particularly apt when running Crack, which gives the administrator access to everyone's password that he/she penetrates.

5.2.1 Detecting Weak Passwords

Running Crack against a systems password file will enable a system administrator to assess how vulnerable the file is to unauthorized users and how well authorized users select secure passwords. Crack is designed to find standard Unix eight-character DES-encrypted passwords by standard guessing techniques.

Crack takes as its input a series of password files and source dictionaries. It merges the dictionaries, turns the password files into a sorted list, and generates lists of possible passwords from the merged dictionary or from information gleaned about users from the password file. It does not attempt to remedy the problem of allowing users to have guessable passwords, and it should NOT be used in place of getting a really good, secure password program replacement.

The instructions provided in the following sections are general in nature because how you configure Crack and how you run it depends on the platforms on which it resides and on the local security requirements established for your site. M&O personnel should be familiar with these tasks to:

1. Configure the Crack shellscript and config.h files based on the README file and on requirements established for your site. See Section "Configuring Crack" below.
2. Run Crack based on requirements established for your site. See "Running Crack," below.
3. Customize the dictionaries. See Section "Creating Dictionaries," below.

5.2.1.1 Configuring Crack

Although Crack should already be configured for your system, the instructions are provided should you have to reconstruct the makefile as a result of file corruption. Crack has two configuration files: the Crack shellscript, which contains all the installation-specific configuration data, and the file Sources/conf.h, which contains configuration options specific to various binary platforms.

1 In the Crack shellscript, edit the CRACK_HOME variable to the correct value. This variable should be set to an absolute path name through which the directory containing Crack may be accessed on ALL machines on which Crack will be run. (Path names relative to username are acceptable as long as you have some sort of csh.)

There is a similar variable, CRACK_OUT, which specifies where Crack should put its output files ~~by~~ default, this is the same as $CRACK_HOME.

2 Edit the file Sources/conf.h and establish which switches to enable. Each #define has a small note explaining its purpose. Portability of certain library functions, should not be a problem.

3 If using Crack -network (see Section Options, below), generate a Scripts/ **network.conf** file. This file contains a list of hostnames to rsh to, what their binary type is (useful when running a network Crack on several different architectures), an estimate of their relative power (take your slowest machine as unary, and measure all others relative to it), and a list of per-host flags to add to those specified on the Crack command line, when calling that host. There is an example of such a file provided in the Scripts directory.

4 To specify a more precise figure as to the relative power of your machines, play with the **command make** tests in the source code directory. This can provide you with the number of fcrypt()s that your machine can do per second, which is a number that you can plug into your **network.conf** as a measure of your machines' power (after rounding the value to an integer).

5.2.1.2 Running Crack

Crack is a self-installing program. Once the necessary configuration options for the Crack shellscript and config.h have been set, the executables are created via **make** by running the Crack shellscript.

Notes for Yellow Pages (NIS) Users:

To get Crack running from a YP password file, the simplest way is to generate a passwd format file by running:-

ypcat passwd > passwd.yp

and then running Crack on this file.

To launch Crack:

1. From /usr/local/solaris **/crack**, at the command line type: **./Crack**
2. For the single platform version:

   ./Crack [options] [bindir] /etc/passwd [...other passwd files]

3. For the network version:

   ./Crack -network [options] /etc/passwd [...other passwd files]

For a brief overview of the [options] available, see Section "Options," below. Section Support Scripts" briefly describes several very useful scripts.

5.2.1.3 Creating Dictionaries

Crack works by making many individual passes over the password entries that you supply to it. Each pass generates password guesses based upon a sequence of rules, supplied to the program by the user. The rules are specified in a simplistic language in the files gecos.rules and dicts.rules, located in the Scripts directory (see Section "Crack Support Scripts," below).

Rules in Scripts/gecos.rules are applied to data generated by Crack from the pw_gecos and pw_gecos entries of the user's password entry. The entire set of rules in gecos.rules is applied to each of these words, which creates many more permutations and combinations, all of which are tested. After a pass has been made over the data based on gecos information, Crack makes further passes over the password data using successive rules from the Scripts/dicts.rules by loading the whole of Dicts/bigdict file into memory, with the rule being applied to each word from that file. This generates a resident dictionary, which is sorted and uniqued so as to prevent wasting time on repetition. After each pass is completed, the memory used by the resident dictionary is freed up, and re-used when the next dictionary is loaded.

Crack creates the Dicts/bigdict dictionary by merging, sorting, and uniq'ing the source dictionaries, which are to be found in the directory DictSrc and which may also be named in the Crack shellscript, via the $STDDICT variable. (The default value of $STDDICT is /usr/dict/words.)

The file DictSrc/bad_pws.dat is a dictionary which is meant to provide many of those common but non-dictionary passwords, such as 12345678 or qwerty.

To create your own dictionary:

1. Copy your dictionary into the DictSrc directory (use compress on it if you wish to save space; Crack will unpack it while generating the big dictionary).

2 Delete the contents of the Dicts directory by running Scripts/spotless. Your new dictionary will be merged in on the next run.

5.2.1.4 Options

-f Runs Crack in foreground mode, i.e., the password cracker is not backgrounded, and messages appear on stdout and stderr as you would expect. This option is only really useful for very small password files, or when you want to put a wrapper script around Crack.

Foreground mode is disabled if you try running Crack-network -f on the command line, because of the insensibility of rshing to several machines in turn, waiting for each one to finish before calling the next. For more information, read the section about Network Cracking without NFS/RFS in the README.NETWORK file.

-v Sets verbose mode, whereby Crack will print every guess it is trying on a per-user basis. This is a very quick way of flooding your filestore, but useful if you think something is going wrong.

-m Sends mail to any user whose password you crack by invoking Scripts/nastygram with their username as an argument. The reason for using the script is so that a degree of flexibility in the format of the mail message is supplied; i.e., you don't have to recompile code in order to change the message.

-nvalue Sets the process to be nice()ed to value, so, for example, the switch -n19 sets the Crack process to run at the lowest priority.

-network Throws Crack into network mode, in which it reads the Scripts/network.conf file, splits its input into chunks which are sized according to the power of the target machine, and calls rsh to run Crack on that machine. Options for Crack running on the target machine may be supplied on the command line (for example, verbose or recover mode), or in the network.conf file if they pertain to specific hosts (e.g., nice() values).

-r<pointfile>

This is only for use when running in recover mode. When a running Crack starts pass 2, it periodically saves its state in a pointfile, with a name of the form Runtime/P.* This file can be used to recover where you were should a host crash. Simply invoke Crack in exactly the same manner as the last time, with the addition of the **-r** switch (for example, **-rRuntime/Pfred12345**). Crack will startup and read the file, and jump to roughly where it left off. If you are cracking a very large password file, this can save a lot of time after a crash.

5.2.1.5 Crack Support Scripts

The Scripts directory contains a small number of support and utility scripts, some of which are designed to help Crack users check their progress. The most useful scripts are briefly described below.

Scripts/shadmrg

This is a small script for merging /etc/passwd and /etc/shadow on System V style shadow password systems. It produces the merged data to stdout, and will need redirecting into a file before Crack can work on it.

Scripts/plaster

This is a simple frontend to the Runtime/D* diefiles that each copy of the password cracker generates. Invoking Scripts/plaster will kill off all copies of the password cracker you are running, over the network or otherwise. Diefiles contain debugging information about the job, and are generated so that all the jobs on the entire network can be called quickly by invoking Scripts/plaster. Diefiles delete themselves after they have been run.

Scripts/status

This script rshes to each machine mentioned in the Scripts/network.conf file, and provides some information about processes and uptime on that machine. This is useful when you want to find out just how well your password crackers are getting on during a Crack -network.

Scripts/{clean,spotless}

These are really just frontends to a makefile. Invoking Scripts/clean tidies up the Crack home directory, and removes probably unwanted files, but leaves the pre-processed dictionary bigdict intact. Scripts/spotless does the same as Scripts/clean but obliterates bigdict and old output files, too, and compresses the feedback files into one.

Scripts/nastygram

This is the shellscript that is invoked by the password cracker to send mail to users who have guessable passwords, if the **-m** option is used. Edit it to suit your system.

Scripts/guess2fbk

This script takes your out* files as arguments and reformats the 'Guessed' lines into a slightly messy feedback file, suitable for storing with the others.

An occasion where this might be useful is when your cracker has guessed many peoples' passwords, and then died for some reason (a crash?) before writing out the guesses to a feedback file. Running Scripts/guess2fbk out* >> Runtime/F.new will save the work that has been done.

5.2.1.6 Checking the Log

Crack loads dictionaries directly into memory, sorts and uniques them, before attempting to use each of the words as a guess for each users' password. If Crack correctly guesses a password, it marks the user as done and does not waste further time on trying to break that user's password.

Once Crack has finished a dictionary pass, it sweeps the list of users looking for the passwords it has cracked. It stores the cracked passwords in both plain text and encrypted forms in a feedback file in the directory **Runtime**. Feedback files have names of the form **Runtime/F***. The purpose of this is so that when it is next invoked, Crack can recognize passwords that it has successfully cracked previously, and filter them from the input to the password cracker. This provides an instant list of crackable users who have not changed their passwords since the last time Crack was run. This list appears in a file with name **out*** in the **$CRACK_OUT** directory, or on **stdout**, if foreground mode (**-f**) is invoked (see Section "Options," above).

Similarly, when a Crack run terminates normally, it writes out to the feedback file all encrypted passwords that it has NOT succeeded in cracking. Crack will then ignore all of these passwords next time you run it.

Obviously, this is not desirable if you frequently change your dictionaries or rules, and so there is a script provided, **Scripts/mrgfbk**, which sorts your feedback files, merges them into one, and optionally removes all traces of "uncrackable" passwords, so that your next Crack run can have a go at passwords it has not succeeded in breaking before.

mrgfbk is invoked automatically if you run **Scripts/spotless** (see Section "Crack Support Scripts," above).

5.2.2 Configuring AnlPasswd

anlpasswd was written by Argonne National Laboratory. There is no install script and installation is by hand. anlpasswd consists of a setuid C program that is used to call the anlpasswd Perl script. The Perl script uses several standard include files that come with Perl and several additional files that are included with anlpasswd. Additionally, a dictionary file is used to match attempted passwords against possible bad passwords that are in the dictionary file.

It is assumed that Perl 5.003 is properly installed in /tools/bin/perl5 for each platform that anlpasswd is to be used on. The binary ypstuff most likely be placed in a NFS shared directory (/tools/bin). The actual Perl program that does the work should be placed in /usr/local/anlpasswd

and chmod to 600. This can't be placed in a NFS directory since /tools/bin/ usually isn't "root equivalent" on all machines and this script should be set to root read only.The Perl includes and dictonary file should also be NFS mounted and placed in /tools/lib/anlpasswd.

The remainder of the installation will need to be completed on each individual machine (moving passwd and yppasswd, creating a SUID program, creating the links, etc.)

The final directory structure should be:

/tools/bin/perl5

/usr/local/anlpasswd/anlpasswd (chmod 600)

passwd link to anlpasswd

yppasswd link to anlpasswd

/usr/bin/passwd.orig original passwd program (chmod 644)

yppasswd.orig original yppasswd program (chmod 644)

passwd suidwrap program (chmod 4111)

yppasswd link to /usr/bin/passwd

/tools/lib/anlpasswd/badpats this directory is the same on all

bigdict machines, text files only

encrypt_passwd

im_prompt2.pl

/tools/bin/ypstuff this binary is arch dependent and is located in {source install dir}/bin.ARCH/ypstuff

The following should be done on each machine (as root):

This assumes that the /tools/bin and /tools/lib directories are already setup as directed above.

```
cd /usr/bin
mv passwd passwd.orig
chmod 644 passwd.orig
mv yppasswd yppasswd.orig
chmod 644 yppasswd.orig
cd /usr/local
mkdir anlpasswd
cd anlpasswd
cp {source install dir}/anlpasswd .
chmod 600 anlpasswd
ln -s anlpasswd passwd
ln -s anlpasswd yppasswd
cd /usr/bin
cp {source install dir}/bin.{ARCH}/suidwrap passwd
chmod 4111 passwd
```

ln -s passwd yppasswd

Below is the original installation instructions:

5.2.2.1 Installing anlpasswd

1. Copy and modify the anlpasswd Perl script. This is located in the "anlpasswd/perl" directory of the distribution, and is called logically enough) "anlpasswd". The configuration section of the

code is located near the beginning of the script, and is labeled "Configs". Here are the lines you need to be concerned with changing:

```
@legal_shells = ('/bin/sh, /bin/csh');
```

This is an array containing the valid shells available on your system. Note that this may not necessarily be the same as the information listed in /etc/shells;

for example, on some machines, /bin/csh and /bin/sh don't have to be listed in /etc/shells. This is not the case with anlpasswd; you should set @legal_shells to contain a list of all valid login shells. If you add new login shells to your systems, you need to update this array.

It is very important that the shells listed here are available on all of the machines on your YP network; otherwise, a user may change his/her shell to one that doesn't exist on one of your machines, and therefore will be unable to log in to that machine.

```
unshift(@INC, "/tools/lib/anlpasswd");
```

These are additional locations for Perl to look for the include files that came with the anlpasswd distribution. These are currently set for our local configuration. If you decide to put the Perl libraries in this distribution (im_prompt2.pl, encrypt_passwd) in locations other than the main Perl include directory, you should add those locations to the include file search path (@INC) as shown above.

Otherwise, you can just delete these lines.

```
# $bigdict = large list of words
$dictdir = "/tools/lib/anlpasswd";    # location of dictionaries
$bigdict = "$dictdir/bigdict";        # large list of words
$ypstuffdir = "/tools/bin";           # location of ypstuff executable
$BADPATS = "$dictdir/badpats";        # location of added bad patterns
```

These are the locations of other files that anlpasswd needs. Change these to reflect the location of your dictionaries, the location of the main dictionary (bigdict.sorted, in this case), and the location of the "ypstuff" program. Again, if you're using this package over a networked system, the dictionaries must be located on some filesystem cross-mounted on all of your machines; otherwise this won't work.

There is a badpats file in the perl directory to use as an example.

Again location ofthe file is up to your customization, but we put it in with the big dictionary.

Finally, edit the definition of the @dictlist array to contain the names of the additional dictionaries (if any) you want to use.

2. Decide where you want to put the anlpasswd script, and copy it there. If you are installing this on a network of machines, this location must be on a filesystem cross-mounted on all machines.

Since Perl disallows running setuid Perl scripts, the anlpasswd script is not executed directly. Instead, a setuid C wrapper is used to call the anlpasswd script. The Perl script should not be

executable or setuid, and should not be located in anyone's path. You should probably "chmod 600 anlpasswd" to be safe.

3. Make a link to anlpasswd called "passwd", i.e.

ln -s anlpasswd passwd

If you are using YP, make another link for yppasswd:

ln -s anlpasswd yppasswd.

4. Decide where you want the passwd executable to reside. This is a C wrapper running setuid to root which calls the anlpasswd script. This will probably be in /bin or /usr/local/bin.

You should keep a copy of the original passwd program around somewhere, in case something breaks, but it should not be executable. A good idea would be to do the following:

cd /usr/bin (or wherever the original passwd program was kept)

mv passwd passwd.orig

chmod 644 passwd.orig

5. Modify anlpasswd/c-routines/suidwrap.c to suit your local configuration. The PASSWD_ACTUAL constant contains the location of the "passwd" link to anlpasswd. Similarly

the YPPASSWD_ACTUAL constant contains the location of the "yppasswd" link. You should change these to reflect the locations you chose in steps 2 and 3. The current settings assume the

yppasswd and passwd executables can be located in /bin or /usr/local/bin. You should modify these to reflect the location(s) you chose in step 4.

6. Run "make" in anlpasswd.ARCH/c-routines. This will compile the suid wrapper and the ypstuff executable. In the sgi directory, manally run the lines in the makefile that are commented out.

Copy "suidwrap" to the location you chose in step 4, and rename this copy as "passwd". (Be sure you saved your original passwd program in a safe place). Change this to be executable by all users, and set it to run suid to root (you must be logged in as root to do this). Make a link to this file called "yppasswd".

i.e., if passwd_exec_dir is the location you chose in step 4,

```
cp suidwrap passwd_exec_dir/passwd
  cd passwd_exec_dir
  chmod 4111 passwd
ln -s passwd yppasswd
```

Unless the location you have chosen for the passwd executable is on a partition cross-mounted on all machines, you will have to repeat this procedure on every machine (or architecture) on your network. (Don't confuse the executable C wrapper with the Perl anlpasswd script; there should only be one copy of the Perl script, on a partition accessible by all the machines on your network.)

7. Copy anlpasswd.ARCH/c-routines/ypstuff to the location you chose for it in step 1.

8. Put the large dictionary file in the location you chose in step 1. There is a C program and instructinos to do this in dictionary-create.

That should be all that is needed to get this program up and runnning. If there are any problems or inaccuracies in this documentation, or have any improvements or bug fixes, please send email to "support@mcs.anl.gov"

5.3 Enforcing Strong Passwords (Secure Shell)

The security risks involved in using ”R”commands such as rlogin, rsh, rexec and rcp are well known but their ease of use has made their use tempting in all but the most secure of environments. Ssh is an easy-to-use, drop in replacement for these commands developed by Tatu Ylonen. Ssh is a ”user”level application. No changes to the host kernel are required. The UNIX server is available as freeware on the Internet but ECS will have a support agreement in place to minimize risk to the program in the event of a product defect.

There are four programs that users use that should be put in /tools/bin for each architecture:

- ssh - replaces rsh, rlogin and rexec for interactive sessions
- scp - replaces rcp for interactive file transfer
- ssh-add - add access to a specific ssh host
- ssh-keygen - generates keys for the local host based on a passcode (long password)

There are three system programs that should be local to each system:

- sshd - the ssh daemon started in /etc/rc2.d NOT in inetd.conf
- ssh-agent - the key management agent started in /etc/rc2.d

- make-ssh-known-hosts - perl script to generate keys for known ssh systems

Several files are generated on installation and when running and are installed locally:

- /etc/ssh_config - system-wide configuration for the ssh client
- /etc/ssh_host_key - contains the long number used for one of the keys
- /etc/ssh_host_key.pub - contains the key known to the public
- /etc/ssh_random_seed - base number used in generating keys
- /etc/sshd.pid - the process number of the sshd currently running
- /etc/sshd_config - defines the local security policy

The amount of disk space that the programs and the configurations files takes is less than 6 MB. NOTE: To compile the source requires about 30MB of disk space.

5.3.1 The SSH Encryption Mechanism[1]

Each host has a host-specific RSA key (normally 1024 bits) used to identify the host. Additionally, when the daemon starts, it generates a server RSA key (normally 768 bits). This key is normally regenerated every hour if it has been used, and is never stored on disk.

Whenever a client connects the daemon, the daemon sends its host and server public keys to the client. The client compares the host key against its own database to verify that it has not changed. The client then generates a 256 bit random number. It encrypts this random number using both the host key and the server key, and sends the encrypted number to the server. Both sides then start to use this random number as a session key which is used to encrypt all further communications in the session. The rest of the session is encrypted using a conventional cipher. Currently, IDEA, DES, 3DES, ARCFOUR, and TSS (a fast home-grown algorithm) are supported. IDEA is used by default. The client selects the encryption algorithm to use from those offered by the server.

Next, the server and the client enter an authentication dialog. The client tries to authenticate itself using .rhosts authentication, .rhosts authentication combined with RSA host authentication, RSA challenge- response authentication, TIS challenge response authentication, or password based authentication.

Rhosts authentication is normally disabled because it is fundamentally insecure, but can be enabled in the server configuration file if desired. System security is not improved unless rshd(8), rlogind(8), rexecd(8), and rexd (8) are disabled (thus completely disabling rlogin(1) and rsh(1) into that machine).

If the client successfully authenticates itself, a dialog for preparing the session is entered. At this time the client may request things like allocating a pseudo-tty, forwarding X11 connections,

---

[1] From the *sshd* man page

forwarding TCP/IP connections, or forwarding the authentication agent connection over the secure channel.

5.3.2 How a User uses Secure Shell

(TBD)

5.3.3 Configuration of Secure Shell

A user can set up one's account manually or by using a script called sshsetup. The only thing you need to know before executing the script or running these commands manually is picking a good

passphrase (since you can and *should* use spaces and multiple words with numbers and misspellings and special characters). Note that passwords are NOT echoed back to the screen. PLEASE DO NOT USE THE PASSWORDS/PASSPHRASES USED HERE OR IN ANY OTHER DOCUMENTATION!

Using the script sshsetup should look like:

```
% sshsetup
```

Use a passphrase of at least 10 characters which should include numbers or special characters and MAY include spaces

```
Initializing random number generator...
Generating p:  .++ (distance 6)
Generating q:  .......++ (distance 110)
Computing the keys...
Testing the keys...
Key generation complete.
Enter file in which to save the key ($HOME/.ssh/identity): [RETURN]
Enter passphrase (empty for no passphrase): litt1e 1amp jumb3d
Enter same passphrase again: litt1e 1amp jumb3d
Your identification has been saved in /home/JohnDoe/.ssh/identity.
Your public key is:
1024 37 [lots of numbers] JohnDoe@theagency.nasa.gov
Your public key has been saved in /home/JohnDoe/.ssh/identity.pub
```

And now you are ready!

NOTE: If you have accounts in the miniDAAC, EDF and the VATC, do sshsetup in EACH environment.

To do the same thing manually requires the following commands:

```
% ssh-keygen
Initializing random number generator...
Generating p:  .++ (distance 6)
Generating q:  .......++ (distance 110)
Computing the keys...
Testing the keys...
Key generation complete.
Enter file in which to save the key ($HOME/.ssh/identity): [RETURN]
Enter passphrase (empty for no passphrase): litt1e 1amp jumb3d
Enter same passphrase again: litt1e 1amp jumb3d
Your identification has been saved in /home/JohnDoe/.ssh/identity.
Your public key is:
1024 37 [lots of numbers] JohnDoe@theagency.nasa.gov
Your public key has been saved in /home/JohnDoe/.ssh/identity.pub
% cd ~/.ssh
% cp identity.pub authorized_keys
% cd
% chmod go-w . .ssh .ssh/authorized_keys
```

Note that the last four commands *are* automatically done in the sshsetup script.

** Changing your passphrase **

To change your passphrase, use the following command:

```
% ssh-keygen -p
Enter file in which the key is ($HOME/.ssh/identity): [RETURN]
Enter old passphrase: litt1e 1amp jumb3d
Key has comment 'JohnDoe@theagency.nasa.gov'
Enter new passphrase (empty for no passphrase): br0wn cow 3ats grass
Enter same passphrase again: br0wn cow 3ats grass
Your identification has been saved with the new passphrase.
```

** Using ssh to login to hosts **

*** The simplest way ***

To login, use the command:

```
% slogin defiant
Enter passphrase for RSA key 'JohnDoe@nevermor': br0wn cow 3ats grass
Last login: Sun Feb 22 06:50:59 1998 from echuser.east.hitc.com
No mail.
%
```

NOTE: The first time you login to a host the following message will pop up asking if you want to continue. In response, type yes and [enter]:

```
Host key not found from the list of known hosts.
Are you sure you want to continue connecting (yes/no)? yes
Host 't1acg01' added to the list of known hosts.
```

To transfer a file, use the command:

```
% scp hostone:/etc/info info
Enter passphrase for RSA key 'bpeters@nevermor':  br0wn cow 3ats grass
stty: : Function not implemented
```

This will copy the file /etc/info from hostone to your local host. Note that your passphrase is needed to initiate the transfer. The "stty: : function not implemented" is a nuisance information message but does not impact the transfer.

IMPORTANT NOTE: The default directory on the *target* host is always the users HOME directory.

Also, one may send/receive file recursively using "-r" such as:

```
% scp -r ~/files/* hostone:~/files
```

will send what is in the home directory files subdirectory to the target host hostone in the home files directory.

To execute a command remotely, use the command:

```
% ssh whoisonfirst "ps -ef"
Enter passphrase for RSA key 'bpeters@nevermor':  br0wn cow 3ats grass
```

*** A layer of convienence ***

If you are already a user of "r" commands, you probably know about the .rhost file. Ssh will allow a user to setup the .rhost equivalent called .shost in one's home directory. .Rhost and .shost contain the names of the hosts to which one normally connects. The nice thing about using it is one need not enter one's passphrase. Unlike "r" commands, however, ssh commands use long strings of numbers to gain access which makes it quite difficult for an intruder to impersonate a legitimate user. One word of caution, however. If you leave your terminal while logged on, a passerby could logon to any host in your .rhost/.shosts file and potentially cause malicious damage to you and your colleagues work. Be aware!

NOTE: ssh checks the mode of .shost, so change permission on .shost by typing:

```
% chmod 600 /home/JohnDoe/.shost
```

where you must substitute your own home directory for /home/JohnDoe.

*** Multiple connections ***

If you open multiple connections, it is more convienient to keep your keys in system memory. To do this requires executing two commands:

```
% ssh-agent tcsh

% ssh-add

Need passphrase for /home/JohnDoe/.ssh/identity (bpeters@nevermor).

Enter passphrase: br0wn cow 3ats grass

Identity added: /home/JohnDoe/.ssh/identity (bpeters@nevermor)
```

You may replace "tcsh" above with your favorite shell (sh, ksh, csh).

Now, one may make connections (slogin, scp, ssh) to hosts that are running ssh without using a passphrase.

** Other notes **

IMPORTANT: Ssh will automatically "tunnel" X sessions without user involvement even through multiple hops. However, it is important to NOT change the DISPLAY parameter or X will not work!

In an effort to reduce the probability of intrusion, ECS does not use the default TCP/IP port for ssh. So if you use ssh from facility other than the DAACs or the Landover facility, call the ECS help desk for the port so that your configuration can be changed.

5.3.4 Administration of Secure Shell

(TBD) See Audit File var/log/ssh

etc/ssh/config

5.4 Monitoring Requests for Network Services (TCP Wrappers)

With TCP Wrapper, you can monitor and filter incoming requests for network services, such as FTP.

TCP Wrapper provides small daemon wrapper programs that can be installed without any changes to existing software or to existing configuration files. The wrappers report the name of the client host and the name of the requested service; the wrappers do not exchange information with the client or server applications, and impose no overhead on the actual conversation between the client and server applications. The usual approach is to run one single daemon process that waits for all kinds of incoming network connections. Whenever a connection is established, this daemon runs the appropriate server program and goes back to sleep, waiting for other connections.

M&O personnel will monitor requests for these network services:

Client	Server	Application
ftp	ftpd	file transfer
finger	fingerd	show users

You will monitor the **log** file. To view log, at the command line type

nowait /var/log/wrappers

Standard Unix commands can be added, such as vi, emacs, or lp -d:

more/var/log/ wrappers | lp -d [printer name]

The log file provides information concerning who tried to access the network service. TCP Wrapper blocks any request made by unauthorized users. TCP Wrapper can be configured to send a message to any administrator whose request is rejected.

5.5 INSTALLATION, CONFIGURATION, and TESTING for Wrappers

Become root on the server that you would like to install wrappers

Note: This documentation is written for sun servers; however, you can replace

the package "wrappers_sun.7.4.tar" for other OS and proceed using the steps below.

(1) Go to any servers with root priveledge

cd /tools/admin/install_pkgs

ftp wrappers_sun.7.4.tar to the server that you would like to install

(put it in /tmp so it does not interfere with others)

(2) cd /etc

cp /etc/inetd.conf /etc/inetd.conf.orig

tar xvf /tmp/wrappers_sun.7.4.tar

(3) Create /etc/hosts.allow if it does not exist. Using the template shown below.

/etc/hosts.allow

==========

Added for HIPPI network. Do Not Delete!

ALL: 192.168.1.: BANNERS /etc/wrappers/banners : ALLOW

ALL: 198.118.: BANNERS /etc/wrappers/banners : ALLOW

ALL: 198.118.192.: BANNERS /etc/wrappers/banners : DENY

ALL: 192.150.28.: BANNERS /etc/wrappers/banners : ALLOW

ALL: 38.177.222.: BANNERS /etc/wrappers/banners : ALLOW

ALL: 128.183.: BANNERS /etc/wrappers/banners : ALLOW

ALL: ALL: BANNERS /etc/wrappers/banners : DENY

(4) Insert the following lines into /etc/inetd.conf (usually right after the current ftp and telnet declaration entries):

ftp stream tcp nowait root /etc/wrappers/tcpd in.ftpd

telnet stream tcp nowait root /etc/wrappers/tcpd in.telnetd

and then comment the existing ftp and telnet entries there.

This is what it should look like:

#ftp stream tcp nowait root /usr/sbin/in.ftpd in.ftpd

#telnet stream tcp nowait root /usr/sbin/in.telnetd in.telnetd

ftp stream tcp nowait root /etc/wrappers/tcpd in.ftpd

telnet stream tcp nowait root /etc/wrappers/tcpd in.telnetd

(5) Make sure these lines are commented out from /etc/inetd.conf (as shown):

###

#

This is the original inetd.conf configuration

#

```
##############################################################################
#shell  stream  tcp     nowait  root    /usr/sbin/in.rshd       in.rshd
#login  stream  tcp     nowait  root    /usr/sbin/in.rlogind    in.rlogind
#exec   stream  tcp     nowait  root    /usr/sbin/in.rexecd     in.rexecd
#comsat dgram   udp     wait    root    /usr/sbin/in.comsat     in.comsat
#talk   dgram   udp     wait    root    /usr/sbin/in.talkd      in.talkd
#
```

(6) Insert this block of lines to /etc/inetd.conf (usually right after the block in step 5):

```
##############################################################################
#
#  This is the setup associated with the configuration of TCP WRAPPERS
#
##############################################################################
shell   stream  tcp     nowait  root    /etc/wrappers/tcpd      in.rshd
login   stream  tcp     nowait  root    /etc/wrappers/tcpd      in.rlogind
exec    stream  tcp     nowait  root    /etc/wrappers/tcpd      in.rexecd
comsat  dgram   udp     wait    root    /usr/sbin/in.comsat     in.comsat
talk    dgram   udp     wait    root    /etc/wrappers/tcpd      in.talkd
```

(6) ps -ef |grep inetd;

kill -HUP "inetd process number"

(7) Test to make sure that you can login from the existing host and also from another host on a different subnet. If it all goes well, that concluded the installation. Otherwise, go back to step (1) and doublecheck your steps.

(8) chmod 755 /etc/hosts.allow

touch /var/log/wrappers

chmod 755 /var/log/wrappers

rm /tmp/wrappers_sun.7.4.tar

rm /etc/inetd.conf.orig

5.6 Monitoring File and Directory Integrity (Tripwire)

Tripwire is a tool that aids in the detection of unauthorized modification of files resident on Unix systems. Tripwire is automatically invoked at system startup. This utility checks file and directory integrity by comparing a designated set of files and directories against information stored in a previously generated database. Tripwire flags and logs any differences, including added or deleted entries. When run against system files regularly, Tripwire spots any changes in critical system files, records these changes into its database, and notifies system administrators of corrupted or tampered files so that they can take damage control measures quickly and effectively. With Tripwire, system administrators can conclude with a high degree of certainty that a given set of files remain free of unauthorized modifications if Tripwire reports no changes.

> **Note:** Since system files should not change and users' files change constantly, Tripwire should be used to **monitor only system files**. The list of system files you want to monitor is stored in **./configs/tw.conf**.

Tripwire is configured to mail the system administrator any output that it generates. However, some files on your system may change during normal operation, and this necessitates updating the Tripwire database.

5.6.1 Installation of Tripwire

Note: This is written for Solaris; however, Tripwire install for other OS (e.g. HP and SGI) is the same.

(1) Go to any servers with root privilege.

cd /tools/admin/install_pkgs

ftp tripwire_sun.1.2.tar to the server that you would like to install

(put it in /tmp so it does not interfere with others)

(2) cd /etc

cp /etc/inet/inetd.conf /etc/inet/inetd.conf.orig

tar xvf /tmp/tripwire_sun.1.2.tar

(3) To setup:

/etc/tripwire-1.2/src/tripwire -init

This will create a database file resided in

/etc/tripwire-1.2/src/databases/tw.db_"whatever_the_hostname"

(4) To test: touch /etc/intruder

/etc/tripwire-1.2/src/tripwire -v > /tmp/tw.report

This should reports the present of /etc/intruder. It works !!! Yeah !!!

(5) rm /etc/intruder

rm /tmp/tw.report

rm /tmp/tripwire_sun.1.2.tar

5.6.2 Updating the Tripwire Database

You can update your Tripwire database in two ways. The first method is interactive, where Tripwire prompts the user whether each changed entry should be updated to reflect the current state of the file, while the second method is a command-line driven mode where specific files/entries are specified at run-time.

5.6.2.1 Updating Tripwire Database in Interactive mode

Running Tripwire in Interactive mode is similar to the Integrity Checking mode. However, when a file or directory is encountered that has been added, deleted, or changed from what was recorded in the database, Tripwire asks the user whether the database entry should be updated.

For example, if Tripwire is run in Interactive mode and a file's timestamp changed, Tripwire will print out what it expected the file to look like, what it actually found, and then prompt the user whether the file should be updated. For example,

```
/etc/hosts equiv
    st_mtime: Wed May  5 15:30:37 1993      Wed May  5 15:24:09 1993
    st_ctime: Wed May  5 15:30:37 1993      Wed May  5 15:24:09 1993
---> File:  /etc/hosts equiv
---> Update entry?  [YN(y)nh?] y
```

You could answer yes or no, where a capital 'Y' or 'N' tells Tripwire use your answer for the rest of the files. (The 'h' and '?' choices give you help and descriptions of the various inode fields.)

While this mode may be the most convenient way of keeping your database up-to-date, it requires that the user be "at the keyboard." A more conventional command-line driven interface exists, and is described next.

5.6.2.2 Updating Tripwire Database in Database Update Mode

Tripwire supports incremental updates of its database on a per-file/directory or tw.config entry basis. Tripwire stores information in the database so it can associate any file in the database with the tw.config entry that generated it when the database was created.

Therefore, if a single file has changed, you can:

```
tripwire -update /etc/newly.installed.file
```

Or, if an entire set of files that made up an entry in the tw.config file changed, you can:

tripwire -update /usr/local/bin/Local_Package_Dir

In either case, Tripwire regenerates the database entries for every specified file. A backup of the old database is created in the **./databases** directory.

Tripwire can handle arbitrary numbers of arguments in Database Update mode.

The script **twdb_check.pl** script is an interim mechanism to ensure database consistency. Namely, when new entries are added to the tw.config file, database entries may no longer be associated with the proper entry number. The twdb_check.pl script analyzes the database, and remaps each database entry with its proper tw.config entry.

5.6.3 Configuring the tw.config file

Edit your **tw.config** file in the **./configs** directory, or whatever filename you defined for the Tripwire configuration file, and add all the directories that contain files that you want monitored. The format of the configuration file is described in its header and in the "man" page. Pay especially close attention to the select-flags and omit-lists, which can significantly reduce the amount of uninteresting output generated by Tripwire. For example, you will probably want to omit files like mount tables that are constantly changed by the operating system.

Run Tripwire with **tripwire -initialize**. This will create a file called **tw.db_[hostname]** in the directory you specified to hold your databases (where [hostname] will be replaced with your machine hostname).

Tripwire will detect changes made to files from this point on. You ***must*** be certain that the system on which you generate the initial database is clean; however, Tripwire cannot detect unauthorized modifications that have already been made. One way to do this would be to take the machine to single-user mode, reinstall all system binaries, and run Tripwire in initialization mode before returning to multi-user operation.

This database must be moved someplace where it cannot be modified. Because data from Tripwire is only as trustworthy as its database, choose this with care. It is recommended to place all the system databases on a read-only disk (you need to be able to change the disk to writeable during initialization and updates, however), or exporting it via read-only NFS from a "secure-server." (This pathname is hardcoded into Tripwire. Any time you change the pathname to the database repository, you must recompile Tripwire. This prevents a malicious intruder from spoofing Tripwire into giving a false "okay" message.)

We also recommend that you make a hardcopy printout of the database contents right away. In the event that you become suspicious of the integrity of the database, you will be able to manually compare information against this hardcopy.

Once you have your database set up, you can run Tripwire in Integrity Checking mode by typing **tripwire** on the command line from the directory in which Tripwire has been installed.

5.7 Reporting Security Breaches

Reporting of Security breaches shall be in accordance with EOSDIS Security Policy and Guides (EOSDIS-IVV-0821.2-9/30/97). Appendix B and C.

5.8 Initiating Recovery from Security Breaches

Recovery from Security breaches shall be in accordance with EOSDIS Security Policy and Guides (EOSDIS-IVV-0821.2-9/30/97). Appendix B and C.

This page intentionally left blank.